socket 编程实验报告

唐宇菲 2018K8009909006

[实验题目]

Socket 应用编程

[实验内容]

HTTP 服务器实验

[实验流程]

1. **预备知识和函数说明** 首先是熟悉了课上新介绍的 Socket API 和 HTTPS 协议所需要的 Openssl 库的接口，并复习了 C 语言中关于文件操作的相关函数，对于打开一个二进制文件，采用 FILE 类型的指针 FILE *fp，和 fopen（）函数，并设置“rb”表示用只读模式打开一个二进制文件，对于一般的文本文件操作，有三组函数(对于文本文件单个字符的读写:fgetc 和 fputc，对于文本文件字符串的读写:fgetc 和 fputs，对于整个文本文件，将其输入或者输出到对应的文本文件中，有类似于输入输出到终端的 scanf 和 prinf，这里采用 fprint 和 fscanf），并且了解到关于文本的随机读写函数，文本内置一个位置指示器，用于指示当前文本的位置，类似于数组下标的作用，用 ftell(fp)表明当前文本文档的位置指示器所在,用 fseek 函数中的原始位置 whence 和偏移量 offset 来指定从某些特定位置开始读写文件。例如 SEEK_END，表示文件的位置指示器指向了所读文本的最后一个字节。SEEK_SET 表示文件的开头,这两者是常用的宏。

2. **熟悉代码框架** 2-socket 文件夹中提供了示例文本的框架，首先需要读懂这个示例框架，从 main 函数开始看，由于文件是 https-server-example.c,所以需要加载并初始化 SSL 库，这里从 34-54 行都已经写好相关的初始化工具、载入证书和密钥等过程，不需要进行修改。

```
34  int main()
35  {
36      // init SSL Library
37      SSL_library_init();
38      OpenSSL_add_all_algorithms();
39      SSL_load_error_strings();
40
41      // enable TLS method
42      const SSL_METHOD *method = TLS_server_method();
43      SSL_CTX *ctx = SSL_CTX_new(method);
44
45      // load certificate and private key
46      if (SSL_CTX_use_certificate_file(ctx, "./keys/cnlab.cert", SSL_FILETYP
47          perror("load cert failed");
48          exit(1);
49      }
50      if (SSL_CTX_use_PrivateKey_file(ctx, "./keys/cnlab.prikey", SSL_FILETY
51          perror("load prikey failed");
52          exit(1);
53      }
54
```

接着是 socket 编程的过程，因为只需要写出服务器端，而服务器端需要经过(1)声明 socket 文件描述符，即声明套接字。(2)定义套接字，绑定相应的 IP 地址和端口号。客户端无此绑定操作，因为通常由操作系统分配完成。（3）监听是否有连接请求 listen。

其中 https 协议对应的端口号为 443，同时由于 socket（）函数中，设置对应的地址族（ipv4 或者 ipv6），规定了固定的数据传输方式(stream 或者 diagram），如果对应的协议只有一种，例如在 ipv4 下面向连接(stream)的协议只有 TCP,在 ipv4 下面对数据报(diagram)

的协议只有 UDP，则对应的协议会直接由操作系统分配生成，所以默认为 0 即可。如果有两种以上的协议，则需要特殊指定。

从套接字的声明、绑定、到监听，步骤如下：

```
55      // init socket, listening to port 443
56      int sock = socket(AF_INET, SOCK_STREAM, 0);
57      if (sock < 0) {
58          perror("Opening socket failed");
59          exit(1);
60      }
61      int enable = 1;
62      if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int)) <
63          perror("setsockopt(SO_REUSEADDR) failed");
64          exit(1);
65      }
66
67      struct sockaddr_in addr;
68      bzero(&addr, sizeof(addr));
69      addr.sin_family = AF_INET;
70      addr.sin_addr.s_addr = INADDR_ANY;
71      addr.sin_port = htons(443);
72
73      if (bind(sock, (struct sockaddr*)&addr, sizeof(addr)) < 0) {
74          perror("Bind failed");
75          exit(1);
76      }
77      listen(sock, 10);
```

其中 61-65 行对应一个超时操作的判断，这些都不需要修改。接下来对应到监听后的 accept 操作，while(1)表示服务器打开，只要没有特殊情况，就不关闭服务器，之前的监听操作开启后，后续一直处于监听状态，如果 accept 失败，则由 83-86，函数的返回值为-1 时，打印错误信息。

```
78
79      while (1) {
80          struct sockaddr_in caddr;
81          socklen_t len;
82          int csock = accept(sock, (struct sockaddr*)&caddr, &len);
83          if (csock < 0) {
84              perror("Accept failed");
85              exit(1);
86          }
87          SSL *ssl = SSL_new(ctx);
88          SSL_set_fd(ssl, csock);
89          handle_https_request(ssl);
90      }
91
92      close(sock);
93      SSL_CTX_free(ctx);
94
95      return 0;
96  }
```

接下来进入到需要进行修改和调整的 handle_https_request 函数。

```
13  void handle_https_request(SSL* ssl)
14  {
15      const char* response="HTTP/1.0 200 OK\r\nContent-Length: 27\r\n\r\nCNL
16      if (SSL_accept(ssl) == -1){
17          perror("SSL_accept failed");
18          exit(1);
19      }
20      else {
21          char buf[1024] = {0};
22          int bytes = SSL_read(ssl, buf, sizeof(buf));
23          if (bytes < 0) {
24              perror("SSL_read failed");
25              exit(1);
26          }
27          SSL_write(ssl, response, strlen(response));
28      }
29      int sock = SSL_get_fd(ssl);
30      SSL_free(ssl);
31      close(sock);
32  }
```

首先 const char*response 是我们最后要输出的目标，对于 GET 反馈的不同情况（文件是否存在于服务器端的文件夹中），输出 200、404 等不同的状态码，这里需要进行修改。同时注意 SSL_accept 和 accept 的区别，第一个是对于客户端发送的连接请求的接收，

SSL_accept（）则是对密钥的确定信息。通过 SSL_read 读入客户端发送的 https 请求到 buf 中，接下来的任务是对 buf 的数据进行截断和解析，并且最终通过 SSL_write 函数发送出相应的 Https 响应(response)，所以对于 443 端口而言，需要修改的地方就是 handle_https_request 中从 21 行 char buf[1024]={0}开始，到 27 行 SSL_write（）截止的内容。

3. 进行正式的实验内容

（i）443 端口 404 Not Found

```
19      else {
20          char buf[1024] = {0};
21          int bytes = SSL_read(ssl, buf, sizeof(buf));
22          if (bytes < 0) {
23              perror("SSL_read failed");
24              exit(1);
25          }
26          //
27          else{
28              char string[1024];
29              int i, k = 0;
30              for( i = 4; buf[i] != ' ';){
31                  string[k] = buf[i];
32                  i++;
33                  k++;
34              }
35              string[k] = '\0';
36
37              FILE *fp;
38              fp = fopen(string,"rb");
39              if(fp == NULL){
40                  const char* response="HTTP/1.0 404 Not Found\r\n";
41                  SSL_write(ssl, response, strlen(response));
42              }
```

这个端口最简单，因为没有发送文件的任务。

首先通过 SSL_read 得到一个 https 请求，并且将其写入 buf 中，返回值 bytes 表示是否接收成功。

27-35 行是读取 buf 中有关于 URL 的信息，因为我们已经指定实验内容中所有 https 请求都是 get，并且根据 https 的请求报文格式，知道 URL 是 GET 空格<URL>空格,所以可以再设置一个新的字符串 string 来接收，最后需要加上‘\0’至此，将 https 请求中的文件地址信息储存在了 string 中。

接下来，就是要确定请求的文件是否在服务器的文件夹中，定义一个 FILE 类型的指针，通过 fopen（）函数来确定是否能够打开成功，打开成功说明文件存在，不成功说明文件不存在。404 对应于文件不存在的情况。rb 表示读取二进制文件，不存在则直接设置 response 信息，并通过 SSL_write()来发送给客户端。

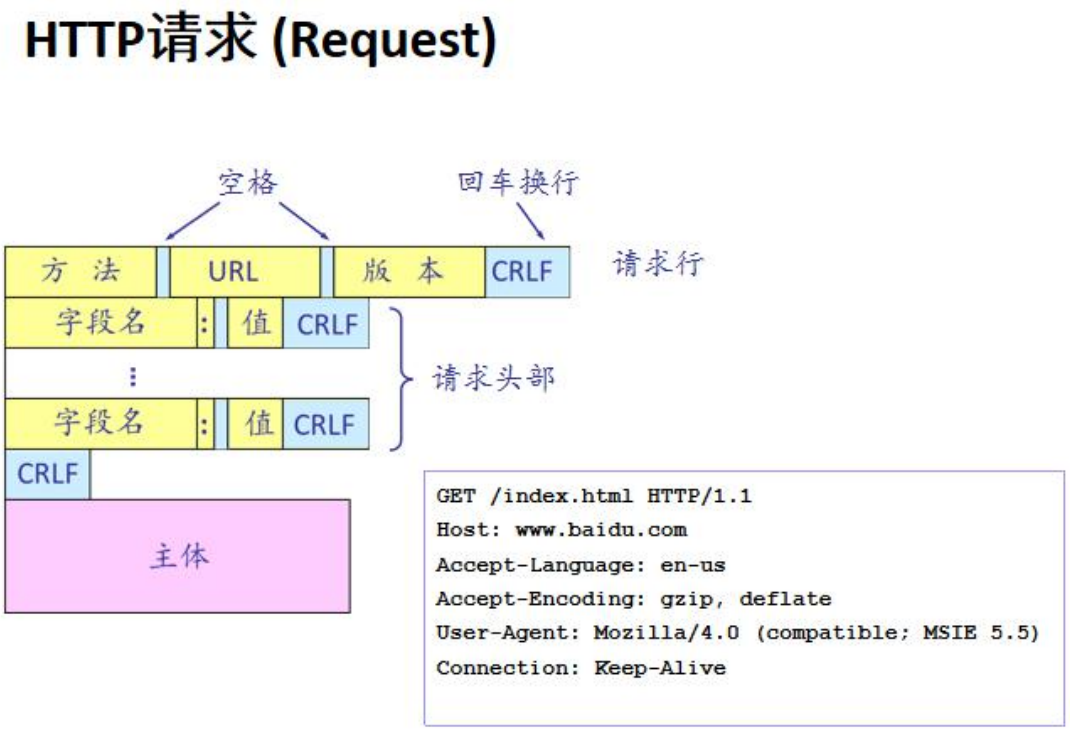
```
FILE *fp;
fp = fopen(string,"rb");
if(fp == NULL){
    const char* response="HTTP/1.0 404 Not Found\r\n";
    SSL_write(ssl, response, strlen(response));
}
```

（ii）443 端口 206 Partial Content

接下来文件成功打开了，说明至少存在或者存在文件的部分。所以接下来需要对 https 的报文请求的字段名进行判断，找到其中是否有 range 信息。

观察请求格式可以发现，首先我们需要跳过第一行，后面得到请求头部，但不清楚具体有几行请求头部，只能通过两个连续的回车换行得到，所以需要设置变量 i 来对 buf 中的信息进行遍历，遍历从第二行

的每个字段名，将其与"Range”进行判断，利用<string.h>库函数中存在的判断字符串是否存在某个子串的 strstr 函数。



设置 find 用来表示，是否找到了这样的匹配字段，如果找到了，则设置 find=1，说明这属于 206 的情况，否则属于 202 的情况。

首先 44 行设置 find 判断是属于哪一种情况，接着设置 l，l 是后续用于处理 partial 的部分信息的文件开始与结束的标记，用于判断最终发送的文件是从哪个字节开始，到哪个字节结束。

然后进行 i 的循环，一直到第一行结束（因为字段名在第二行开始），最后再 i++，表明从第二行开始读取字段名。

```
43          else{
44              int find = 0;
45              int l;
46              for(i = 0; buf[i]!='\n';i++);
47              i++;
48              while(!(buf[i]=='\r' && buf[i + 1]=='\n' && buf[i + 2]=='
49                  char lines[1024];
50                  for(k = 0; buf[i]!=':';k++,i++){
51                      lines[k] = buf[i];
52                  }
53                  char buf_206_range[1024];
54                  l = 0;
55                  for(; buf[i]!='\n';i++){
56                      buf_206_range[l] = buf[i];
57                      l++;
58                  }
59                  buf_206_range[l]='\0';
60                  if(strstr(lines,"Range")!=NULL){
61                      find = 1;
62                      break;
63                  }
64              }
65
```

49-52 行表明，我们读取了到冒号之前的内容，进入新定义的字符串 lines，这就完成了第一个字段名的录入。接下来需要考虑，如果读到 Range：后面的值应该是 100-200，或者 bytes=100-200，或者 100 一类似的这三种情况，所以我们需要设置一个新的字符串来存储字段名后定义的值：buf_206_range，它需要用 l 进行赋值和遍历，从与字段名同一行的：之后的第一个字符开始，到最后的换行符结束。（其实大多数情况下，这个值并不是 Range 字段对应的值。）

53-59 行完成对于字段名所对应的值的录入。

60-64 行进行字符串的比较操作，判断 lines 中是否含有 " Range " 作为字串，如果有，则设置 find=1，标记找到，并退出循环。否则继续循环直到达到 https 报文请求的末尾。

```
66              if(find == 1){
67                  int start,end;
68                  for(i = 0; !isdigit(buf_206_range[i]);i++);
69                  start = atoi(&buf_206_range[i]);
70
71                  int len;
72                  for(i = 0; buf_206_range[i] !='-';i++);
73                  if( isdigit(buf_206_range[i + 1])){
74                      end = atoi(&buf_206_range[i + 1]);
75                      len = end - start;
76                  }
77                  else{
78                      fseek(fp,0,SEEK_END);
79                      end = ftell(fp);
80                      len = end - start;
81                  }
82                  char* response;
83                  response = (char *)malloc(len*sizeof(char) + 1024);
84                  sprintf(response,"HTTP/1.0 206 Partial Content\r\nCont
85
86                  fseek(fp,start,SEEK_SET);
87                  int compare_len;
88                  while(feof(fp)!= 0 &&  len > 0){
89                      char buf_206[1024];
90                      compare_len = len > 1024? 1024 : len;
91                      fread(buf_206,1,compare_len,fp);
92                      strcat(response,buf_206);
93                      len = len - compare_len;
94                  }
95                  SSL_write(ssl, response, strlen(response));
96              }
```

接下来，如果找到了，即 find=1 时，我们需要确定读出哪部分文件，由于 buf_206_range 字符串中存储了形如“100-200”，或者“butes = 100-200”，或者 " 100- " 的信息，所以需要先跳过开始字节（数字）前面所有的非数字部分，即跳过 bytes=这样的非数字字符，利用 isdigit（）函数即可判断。67 行设置了开始和结束。

68-69 行对于开始位置进行从字符串到数字的转换。由于 atoi 函数自动到非数字字符停下，所以不需要再对 start 的转换进行更多的判断。而文件的结束标志 end 需要先找到’-’符号，然后进行数据的转换，最后确定读取文件的长度 len=end-start。注意到，如果’-’

后面不是数字,说明此时只有开始标识,结束标志默认是文件的结尾,所以为了确定文件的结尾，利用 fseek 和 ftell 结合的方法。

第 79 行先将文件的位置指示器设置到末尾，然后用 ftell 得到这个数据，赋值给 end，同样得到开始到结束的距离。

82-84 行本来在原始的框架中位于最前面的部分，但由于我们需要知道发送数据的长度，直到刚才进行了两次 atoi 的数据转换和 len 的赋值之后，我们才知道需要发送的数据大小，所以这里给出 response 的初始声明，并用 sprintf 将文本的大小打印到字符串中。

下面要来读取文件内容，可以设置 buf_206，因为默认每一次到换行或者文件结束标志 eof 停下,所以 buf_206 的大小并不需要特殊说明，但我们读取的内容至多为 len，所以每读取一次，对 len 进行递减，最后当 len <= 0 时标记读取结束。如果 buf_206 的大小大于需要读取的长度，则以 len 为准，所以 90 行进行了一个大小判断。最后 95 行 SSL_write 写入需要的响应信息。

（iii）443 端口 200 OK

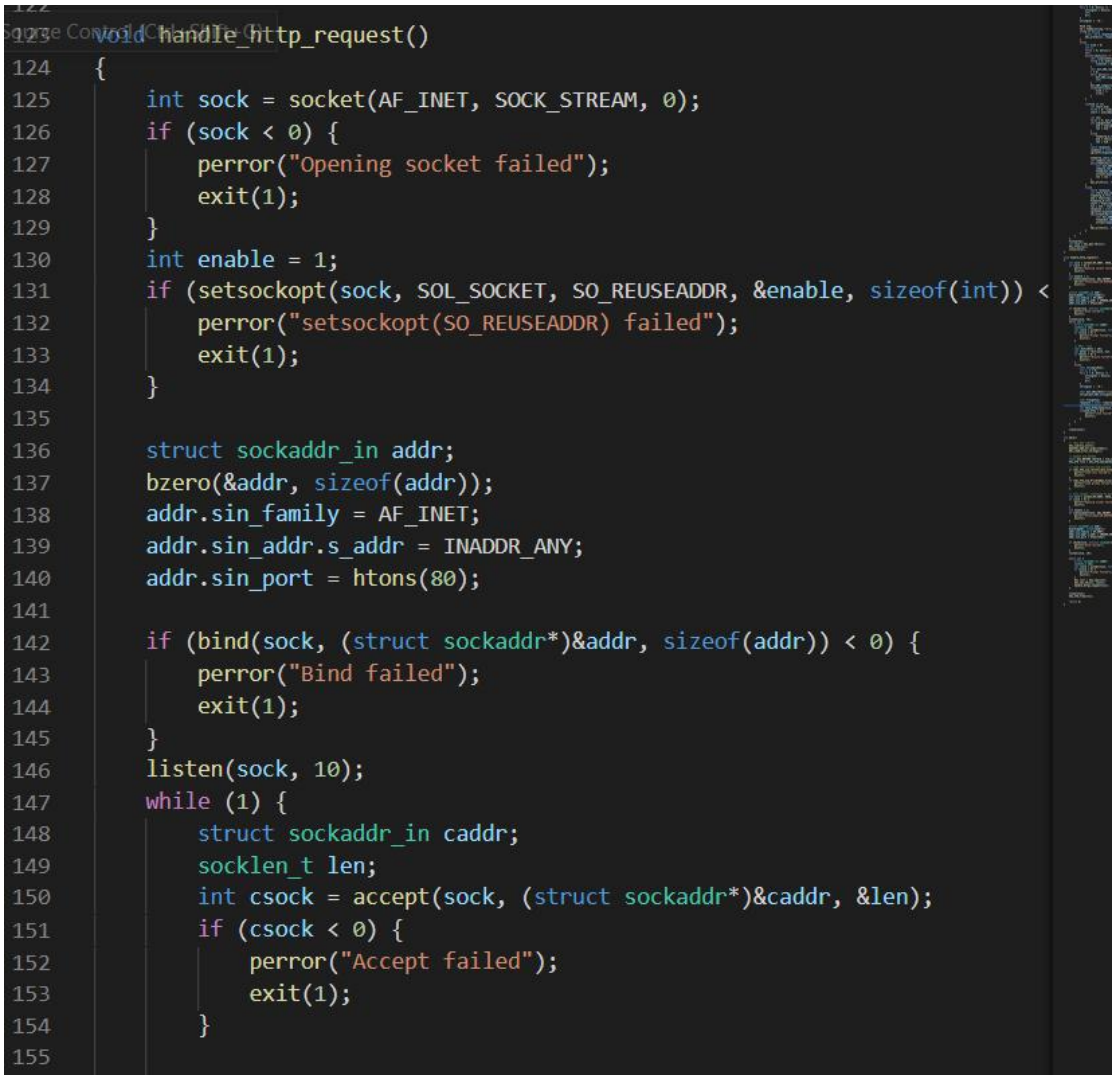
这个条件比 206 判断要简单，步骤类似，需要知道整个文本的开始和结束，所以调用 fseek 和 ftell 的组合，fread 出全部的信息，与需要发送的响应行进行拼接，strcat，最终采用 SSL_write 回复信息。

```
97      else{
98          char* response;
99          int start,end,len;
100         fseek(fp,0,SEEK_END);
101         end = ftell(fp);
102         fseek(fp,0,SEEK_SET);
103         start = ftell(fp);
104         len = end - start;
105         response = (char *)malloc(len*sizeof(char) + 1024);
106         sprintf(response,"HTTP/1.0 200 Partial Content\r\nContent-Length:
107         while(feof(fp)!= 0){
108             char buf_200[1024];
109             fread(buf_200,1,1024,fp);
110             strcat(response,buf_200);
111         }
112         SSL_write(ssl, response, strlen(response));
113     }
114 }
```

（iv）80 端口，301 Moved Permanently

重定向到 https 开头的网址，由于 http 不涉及 SSL，所以直接按照 socket 封装的 API 来进行操作，

套接字声明，定义 socket 文件描述符—bing 套接字定义—超时判断—监听等与框架代码中 main 函数提供的一致，所以 123-155 行不需要修改。



```
123  void handle_http_request()
124  {
125      int sock = socket(AF_INET, SOCK_STREAM, 0);
126      if (sock < 0) {
127          perror("Opening socket failed");
128          exit(1);
129      }
130      int enable = 1;
131      if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int)) <
132          perror("setsockopt(SO_REUSEADDR) failed");
133          exit(1);
134      }
135
136      struct sockaddr_in addr;
137      bzero(&addr, sizeof(addr));
138      addr.sin_family = AF_INET;
139      addr.sin_addr.s_addr = INADDR_ANY;
140      addr.sin_port = htons(80);
141
142      if (bind(sock, (struct sockaddr*)&addr, sizeof(addr)) < 0) {
143          perror("Bind failed");
144          exit(1);
145      }
146      listen(sock, 10);
147      while (1) {
148          struct sockaddr_in caddr;
149          socklen_t len;
150          int csock = accept(sock, (struct sockaddr*)&caddr, &len);
151          if (csock < 0) {
152              perror("Accept failed");
153              exit(1);
154          }
155
```

接下来需要对接收到的信息进行判断，从 157-188 行进行修改。依然用 buf 接收请求信息，这里采用 recv 来收取。如果返回值为-1，表明接收失败，退出。

164-171 行对 http 请求信息的 URL 进行收取。173-174 行实现将以 http 开头的 URL 改为以 http 开头的 URL。176-185 行进行 response 的设置和发送处理。

```
156         // 这里开始添加
157         char buf[1024] = {0};
158         int bytes = recv(sock, buf, sizeof(buf),0);
159         if (bytes < 0) {
160             perror("receive failed");
161             exit(1);
162         }
163         else{
164             char string[1024];
165             int i, k = 0;
166             for( i = 4; buf[i] != ' ';){
167                 string[k] = buf[i];
168                 i++;
169                 k++;
170             }
171             string[k] = '\0';
172 
173             char buf_301[1024]="Location:https";
174             strcat(buf_301,string[4]);
175 
176             char *response;
177             response = (char *)malloc(1024*2 + strlen(string));
178             sprintf(response,"HTTP/1.0 301 301 Moved Permanently\r\n");
179             int send_http=send(sock,response,strlen(response),0);
180             if(send_http < 0){
181                 perror("send failed");
182                 exit(1);
183             }
184         }
185     }
186 
187     close(sock);
188 }
```

[实验结果及分析]

1. 写完上述基本流程中的代码后，通过不断添加 printf 打印相关信息，发现了代码中的如下 6 个错误：

1. fclose（fp)不能关闭空指针

2. feof(fp)==0 表示没有结束

3. 寻找本地文件不能加/ string 从 5 号开始

4. char buf_206 在栈里需要初始化成 0,否则文件找不到末尾-> char buf_206[1024]={0}

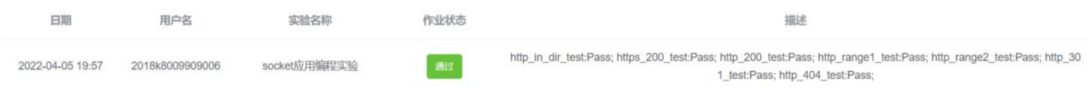
5. while 判断的时候以三个进行判断，因为出现'\n'的时候已经是文件的结尾了，停下的标准是'\r'

6.http 传输时，必须要写长度统计，即使为 0，最后是’\r’’\n’结尾。

在 Linux 上调试结果如下：

```
root@ucas-cod-2022:/home/ucas/Desktop/socket# python3 test/test.py
/usr/lib/python3/dist-packages/urllib3/connectionpool.py:999: InsecureRequestWarning: Unverified HTTPS request is being made to host '10.0.0.1'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  warnings.warn(
/usr/lib/python3/dist-packages/urllib3/connectionpool.py:999: InsecureRequestWarning: Unverified HTTPS request is being made to host '10.0.0.1'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  warnings.warn(
/usr/lib/python3/dist-packages/urllib3/connectionpool.py:999: InsecureRequestWarning: Unverified HTTPS request is being made to host '10.0.0.1'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  warnings.warn(
/usr/lib/python3/dist-packages/urllib3/connectionpool.py:999: InsecureRequestWarning: Unverified HTTPS request is being made to host '10.0.0.1'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  warnings.warn(
/usr/lib/python3/dist-packages/urllib3/connectionpool.py:999: InsecureRequestWarning: Unverified HTTPS request is being made to host '10.0.0.1'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  warnings.warn(
/usr/lib/python3/dist-packages/urllib3/connectionpool.py:999: InsecureRequestWarning: Unverified HTTPS request is being made to host '10.0.0.1'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  warnings.warn(
root@ucas-cod-2022:/home/ucas/Desktop/socket#
```

在 oj 上测试如下：



日期	用户名	实验名称	作业状态	描述
2022-04-05 19:57	2018k8009909006	socket应用编程实验	通过	http_in_dir_test:Pass; https_200_test:Pass; http_200_test:Pass; http_range1_test:Pass; http_range2_test:Pass; http_301_test:Pass; http_404_test:Pass;

最终的代码如下：

```c
#include <errno.h>
#include <unistd.h>
#include <malloc.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <resolv.h>
#include <ctype.h>
#include "openssl/ssl.h"
#include "openssl/err.h"

void handle_https_request(SSL* ssl)
{
    FILE *fp;
    if (SSL_accept(ssl) == -1){
        perror("SSL_accept failed");
        exit(1);
    }
    else{
        char buf[1024] = {0};
        int bytes = SSL_read(ssl, buf, sizeof(buf));
        if (bytes < 0) {
            perror("SSL_read failed");
            perror("SSL_read failed");
            exit(1);
        }
        //
        else{
            char string[1024];
            int i, k = 0;
            for( i = 5; buf[i] != ' ';){
                string[k] = buf[i];
                i++;
                k++;
            }
            string[k] = '\0';
```

```
            fp = fopen(string,"rb");
            if(fp == NULL){
                const char* response="HTTP/1.0 404 Not Found\r\n";
                SSL_write(ssl, response, strlen(response));
            }
            else{
                int find = 0;
                int l;
                for(i = 0; buf[i]!='\n';i++);
                i++;
                char lines[1024];
                char buf_206_range[1024];
                while( !(buf[i]=='\r' && buf[i + 1]=='\n' && buf[i + 2]=='\r' && buf[i+ 3] == '\n')){
                    for(k = 0; buf[i]!=':';k++,i++){
                        lines[k] = buf[i];
                    }
                    l = 0;
                    for(; buf[i]!='\r';i++){
                        buf_206_range[l] = buf[i];
                        l++;
                    }
                    buf_206_range[l]='\0';
                    if(strstr(lines,"Range")!=NULL){
                        find = 1;
                        break;
                    }
                }

                if(find == 1){
                    int start,end;
                    for(i = 0; isdigit(buf_206_range[i]) == 0;i++);
                    start = atoi(&buf_206_range[i]);

                    long int len;
                    for(i = 0; buf_206_range[i] !='-';i++);
                    if( isdigit(buf_206_range[i + 1]) != 0){
                        end = atoi(&buf_206_range[i + 1]);
                        len = end - start + 1;
```

```
77
78                  }
79                  else{
80                      fseek(fp,0,SEEK_END);
81                      end = ftell(fp);
82                      len = end - start + 1;
83                  }
84
85                  char* response;
86                  response = (char *)malloc(len*sizeof(char) + 1024);
87                  sprintf(response,"HTTP/1.0 206 Partial Content\r\nContent-Length: %ld\r\n\r\n",len);
88
89                  fseek(fp,start,SEEK_SET);
90                  int compare_len;
91                  while(feof(fp)== 0 &&  len > 0){
92                      char buf_206[1025]={0};
93                      compare_len = len > 1024? 1024 : len;
94                      fread(buf_206,1,compare_len,fp);
95                      strcat(response,buf_206);
96                      len = len - compare_len;
97                  }
98                  SSL_write(ssl, response, strlen(response));
99              }
100             else{
101                 char* response;
102                 int start,end;
103                 long int len;
104                 fseek(fp,0,SEEK_END);
105                 end = ftell(fp);
106                 fseek(fp,0,SEEK_SET);
107                 start = ftell(fp);
108                 len = end - start;
109                 response = (char *)malloc(len*sizeof(char) + 1024);
110                 sprintf(response,"HTTP/1.0 200 Partial Content\r\nContent-Length: %ld\r\n\r\n",len);
111                 while(feof(fp) == 0){
112                     char buf_200[1025] = {0};
113                     fread(buf_200,1,1024,fp);
114                     strcat(response,buf_200);
```

```
                    }
                    SSL_write(ssl, response, strlen(response));
                }
                fclose(fp);
            }
        }
    }

    int sock = SSL_get_fd(ssl);
    SSL_free(ssl);
    close(sock);
}

void handle_http_request(void)
{
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("Opening socket failed");
        exit(1);
    }
    int enable = 1;
    if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int)) < 0) {
        perror("setsockopt(SO_REUSEADDR) failed");
        exit(1);
    }

    struct sockaddr_in addr;
    bzero(&addr, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = INADDR_ANY;
    addr.sin_port = htons(80);

    if (bind(sock, (struct sockaddr*)&addr, sizeof(addr)) < 0) {
        perror("Bind failed");
        exit(1);
    }
    listen(sock, 10);
    while (1) {
```

```
153         struct sockaddr_in caddr;
154         socklen_t len;
155         int csock = accept(sock, (struct sockaddr*)&caddr, &len);
156         if (csock < 0) {
157             perror("Accept failed");
158             exit(1);
159         }
160         // 这里开始添加
161         char buf[1024] = {0};
162         int bytes = recv(csock, buf, sizeof(buf),0);
163         if (bytes < 0) {
164             perror("receive failed");
165             exit(1);
166         }
167         else{
168             char string[1024];
169             int i, k = 0;
170             for( i = 5; buf[i] != ' ';){
171                 string[k] = buf[i];
172                 i++;
173                 k++;
174             }
175             string[k] = '\0';
176 
177             char buf_301[1024]="Location: https://10.0.0.1/";
178             strcat(buf_301,string);
179 
180             char *response;
181             response = (char *)malloc(1024*2 + strlen(string));
182             char buf_301_format[1024]="\r\n\r\n";
183             sprintf(response,"HTTP/1.0 301 Moved Permanently\r\nContent-Length: 0\r\n");
184             strcat(response,buf_301);
185             strcat(response,buf_301_format);
186             int send_http=send(csock,response,strlen(response),0);
187             if(send_http < 0){
188                 perror("send failed");
189                 exit(1);
190             }
```

```
191         }
192     }
193     close(sock);
194 }
195
196 void https(void)
197 {
198     // init SSL Library
199     SSL_library_init();
200     OpenSSL_add_all_algorithms();
201     SSL_load_error_strings();
202
203     // enable TLS method
204     const SSL_METHOD *method = TLS_server_method();
205     SSL_CTX *ctx = SSL_CTX_new(method);
206
207     // load certificate and private key
208     if (SSL_CTX_use_certificate_file(ctx, "./keys/cnlab.cert", SSL_FILETYPE_PEM) <= 0) {
209         perror("load cert failed");
210         exit(1);
211     }
212     if (SSL_CTX_use_PrivateKey_file(ctx, "./keys/cnlab.prikey", SSL_FILETYPE_PEM) <= 0) {
213         perror("load prikey failed");
214         exit(1);
215     }
216
217     // init socket, listening to port 443
218     int sock = socket(AF_INET, SOCK_STREAM, 0);
219     if (sock < 0) {
220         perror("Opening socket failed");
221         exit(1);
222     }
223     int enable = 1;
224     if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int)) < 0) {
225         perror("setsockopt(SO_REUSEADDR) failed");
226         exit(1);
227     }
228
```

```
    struct sockaddr_in addr;
    bzero(&addr, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = INADDR_ANY;
    addr.sin_port = htons(443);

    if (bind(sock, (struct sockaddr*)&addr, sizeof(addr)) < 0) {
        perror("Bind failed");
        exit(1);
    }
    listen(sock, 10);

    while (1) {
        struct sockaddr_in caddr;
        socklen_t len;
        int csock = accept(sock, (struct sockaddr*)&caddr, &len);
        if (csock < 0) {
            perror("Accept failed");
            exit(1);
        }
        SSL *ssl = SSL_new(ctx);
        SSL_set_fd(ssl, csock);
        handle_https_request(ssl);
    }

    close(sock);
    SSL_CTX_free(ctx);
}


int main(){
    pthread_t thread443,thread80;
    pthread_create(&thread443,NULL,https,NULL);
    pthread_create(&thread80,NULL,handle_http_request,NULL);
    pthread_join(thread443,NULL);
    pthread_join(thread80,NULL);
    return 0;
}
```